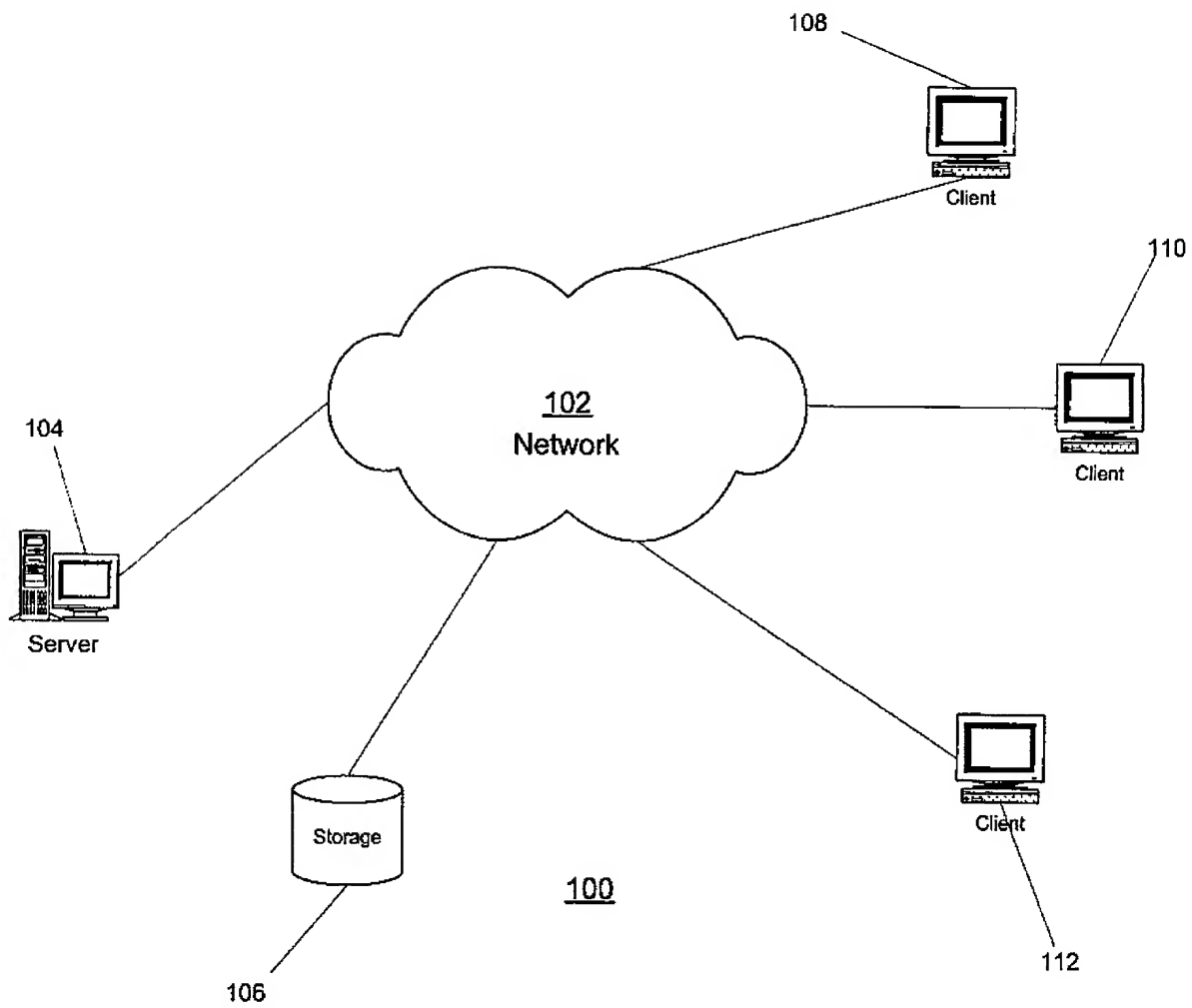
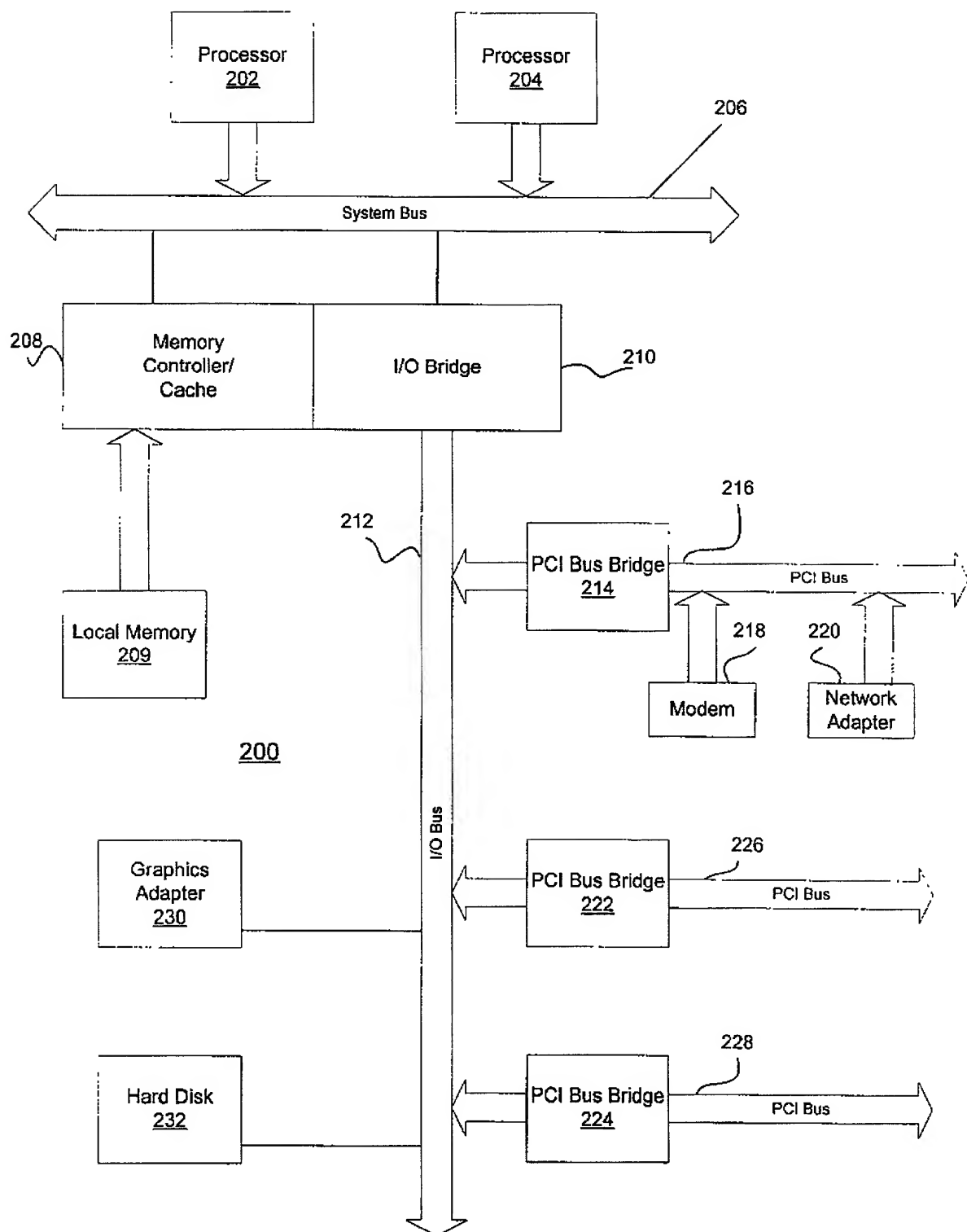


# Figure 1

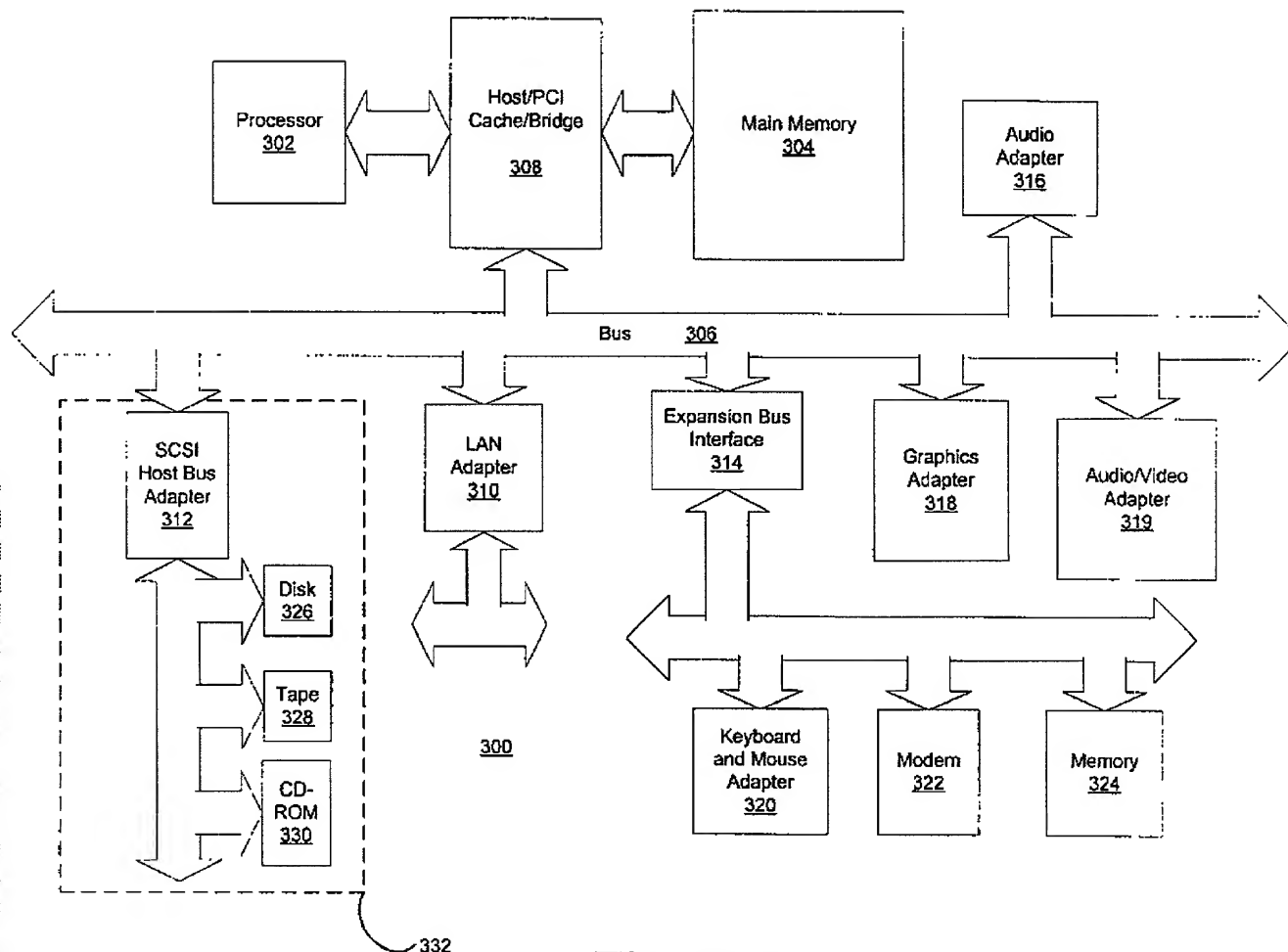
Koved et al.  
 AUS920010942US1  
 Method and Apparatus for Implementing  
 Permission Based Access Control  
 Through Permission Type Inheritance  
 Page 1 of 13





**Figure 2**

FIG. 3 is a block diagram of a computer system 300. The system includes a Processor 302, Host/PCI Cache/Bridge 308, Main Memory 304, and Audio Adapter 316. A Bus 306 connects these components. The Bus 306 also connects to a SCS Host Bus Adapter 312, LAN Adapter 310, Expansion Bus Interface 314, Graphics Adapter 318, and Audio/Video Adapter 319. The SCS Host Bus Adapter 312 is connected to Disk 326, Tape 328, and CD-ROM 330. The Expansion Bus Interface 314 is connected to Keyboard and Mouse Adapter 320, Modem 322, and Memory 324. The system is labeled 300 and 332.



**Figure 3**

Koved et al.  
AUS920010942US1  
Method and Apparatus for Implementing  
Permission Based Access Control  
Through Permission Type Inheritance  
Page 3 of 13

FIG. 4 is a block diagram of a system for implementing permission based access control through permission type inheritance. The system includes a Server 415 and a Client Device 470. The Server 415 contains a Web Page 410 and an Applet 420. The Client Device 470 contains a Web Browser 470, which includes a Bytecode Verifier 440, an Applet Class Loader 445, an Applet Class 450, a Namespace 460, a Security Manager 480, and an Access Controller 485. The Web Page 410 is transmitted to the Bytecode Verifier 440. The Bytecode Verifier 440 transmits data to the Applet Class Loader 445, which then transmits data to the Applet Class 450. The Applet Class 450 is contained within the Namespace 460. The Namespace 460 transmits data to the Security Manager 480. The Security Manager 480 transmits data to the Access Controller 485. The Access Controller 485 transmits data to the Security Manager 480. The Security Manager 480 and the Access Controller 485 are both connected to the Java Virtual Machine 470.

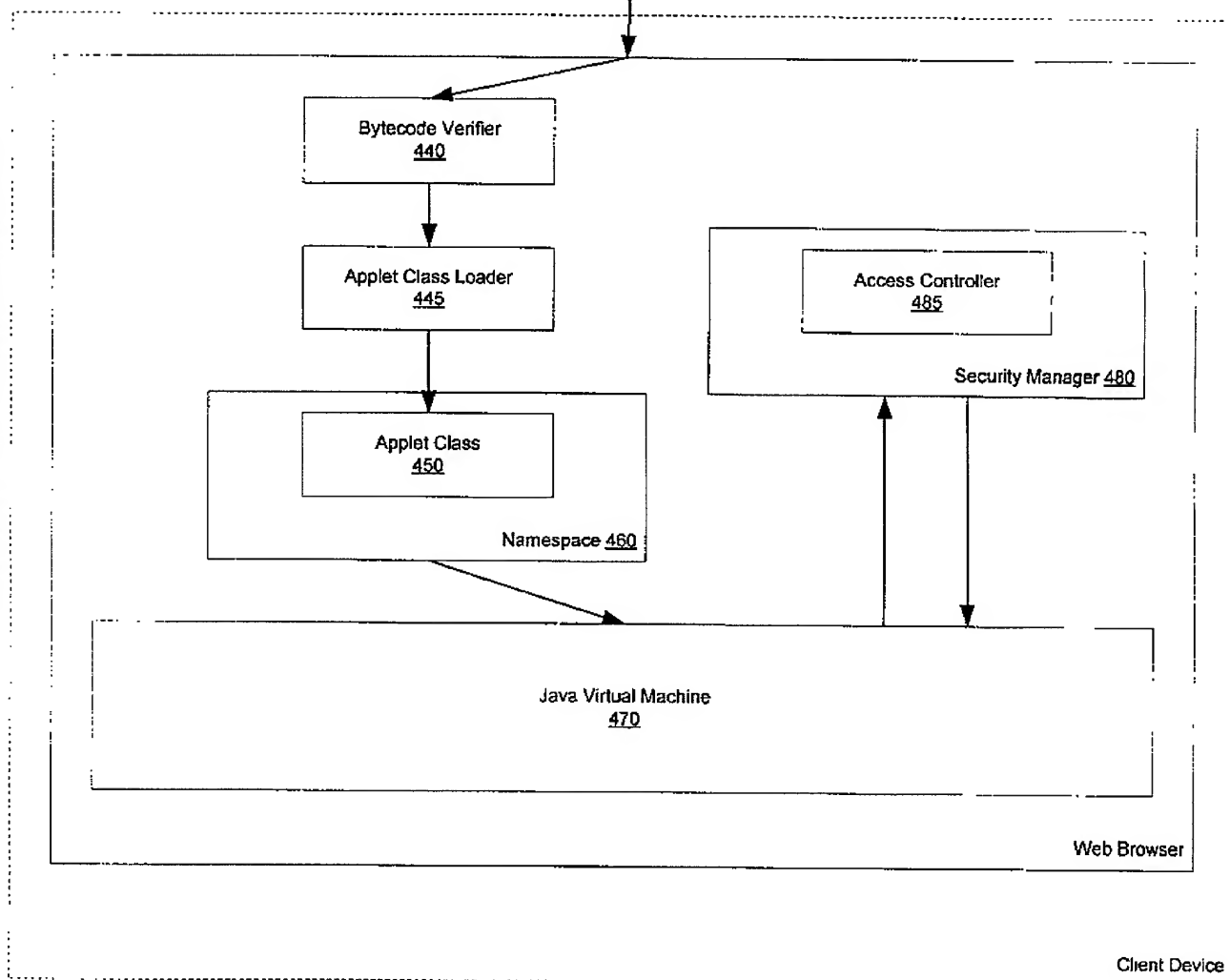
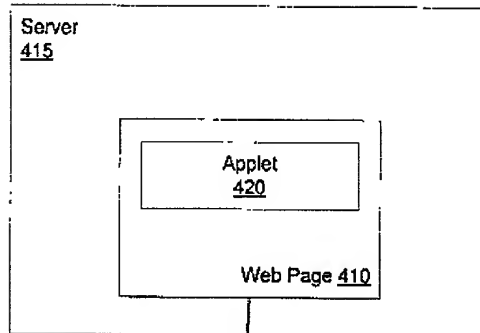


Figure 4

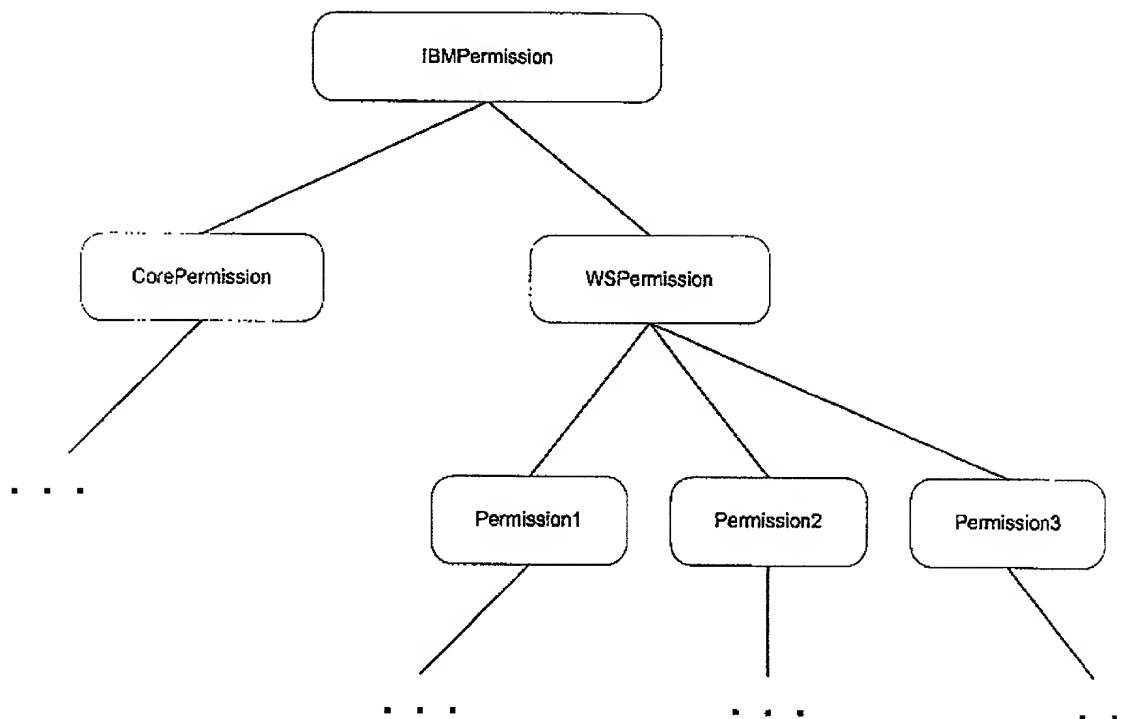


Figure 5

Koved et al.  
AUS920010942US1  
Method and Apparatus for Implementing  
Permission Based Access Control  
Through Permission Type Inheritance  
Page 5 of 13



```

import java.security.BasicPermission;
import java.security.Permission;
import java.security.PermissionCollection;
import java.util.Hashtable;
import java.util.Enumeration;

public class IBMPermission extends BasicPermission
{
    public IBMPermission()
    {
        super("");
        System.out.println("Constructor IBMPermission() called");
    }
    public IBMPermission(String target)
    {
        super(target);
        System.out.println("Constructor IBMPermission(target) called");
    }

    public IBMPermission(String target, String actions)
    {
        super(target, actions);
        System.out.println("Constructor IBMPermission(target, actions) called");
    }
    public boolean implies(Permission perm)
    {
        System.out.println("IBMPermission.implies() called");

        if (perm instanceof IBMPermission)
            return true;
        return false;
    }
    public PermissionCollection newPermissionCollection()
    {
        return new IBMPermissionCollection();
    }
}

```

## Figure 7A

Koved et al.  
 AUS920010942US1  
 Method and Apparatus for Implementing  
 Permission Based Access Control  
 Through Permission Type Inheritance  
 Page 7 of 13

```

final class IBMPermissionCollection extends PermissionCollection
    implements java.io.Serializable
{
    private Hashtable permissions;

    public IBMPermissionCollection()
    {
        permissions = new Hashtable();
    }

    public void add(Permission permission)
    {
        if (!(permission instanceof IBMPermission))
            throw new IllegalArgumentException("Invalid Permission: " +
                                             permission);

        IBMPermission ibmp = (IBMPermission) permission;
        permissions.put(ibmp.getName(), permission);
    }

    public boolean implies(Permission permission)
    {
        if (!(permission instanceof IBMPermission))
            return false;

        System.out.println("permission instanceof IBMPermission == true");

        IBMPermission ibmp = (IBMPermission) permission;
        String permName = ibmp.getName();
        Permission x = (Permission) permissions.get(permName);

        if (x != null)
        {
            System.out.println("We have a direct hit! " + x.getName());
            return x.implies(permission);
        }

        Enumeration permEnum = permissions.elements();

        while (permEnum.hasMoreElements())
        {
            x = (IBMPermission) permEnum.nextElement();
            System.out.println(x.getName());

            if (x.implies(permission))
                return true;
        }

        return false;
    }

    public Enumeration elements()
    {
        return permissions.elements();
    }
}

```

## Figure 7B

Koved et al.  
 AUS920010942US1  
 Method and Apparatus for Implementing  
 Permission Based Access Control  
 Through Permission Type Inheritance  
 Page 8 of 13



```

import java.security.PermissionCollection;
import java.security.AccessController;
import java.security.AccessControlContext;
import java.security.AccessControlException;

public class WSPermission extends IBMPermission
{
    public WSPermission(String target)
    {
        super(target);
        System.out.println("Constructor WSPermission(target) called");
    }

    public WSPermission(String target, String actions)
    {
        super(target, actions);
        System.out.println("Constructor WSPermission(target, actions) called");
    }

    public WSPermission()
    {
        super("");
        System.out.println("Constructor WSPermission() called");
    }

    /**
     * Returns a new IBMPermissionCollection object for storing IBMPermission
     * objects.
     * <p>
     * An IBMPermissionCollection stores a collection of
     * IBMPermission permissions.
     * <p>
     * IBMPermission objects must be stored in a manner that allows them
     * to be inserted in any order, but that also enables the
     * PermissionCollection <code>implies</code> method
     * to be implemented in an efficient (and consistent) manner.
     *
     * @return a new IBMPermissionCollection object suitable for
     *         storing IBMPermission's.
     */
    public PermissionCollection newPermissionCollection()
    {
        System.out.println("newPermissionCollection() was called");
        IBMPermissionCollection ibmPC = new IBMPermissionCollection();

        // the code here checks if an IBMPermissionCollection has been granted.
        // If yes, then the PermissionCollection returned by this
        // method should contain a WSPermission.

        AccessControlContext acc = AccessController.getContext();

        try
        {
            acc.checkPermission(new IBMPermission("PermissionTest"));
            ibmPC.add(new WSPermission("PermissionTest"));
        }
        catch (AccessControlException ace)
        {
            System.out.println("IBMPermission WAS NOT GRANTED");
        }
        return ibmPC;
    }
}

```

## Figure 7C

Koved et al.  
 AUS920010942US1  
 Method and Apparatus for Implementing  
 Permission Based Access Control  
 Through Permission Type Inheritance  
 Page 9 of 13

```

import java.io.*;

public class PermissionTest
{
    public static void main(String args[])
    {
        try
        {
            SecurityManager sm = System.getSecurityManager();

            if (sm != null)
            {
                System.out.println("SecurityManager is checking for " +
                                   "WSPermission");

                sm.checkPermission(new WSPermission("PermissionTest"));
            }

            System.out.println("WSPermission was granted. " +
                               "Permission testing
worked.\n\n\n");

            File inputFile = new File("C:\\winzip.log");
            FileInputStream fis = new FileInputStream(inputFile);
            InputStreamReader isr = new InputStreamReader(fis);
            BufferedReader br = new BufferedReader(isr);

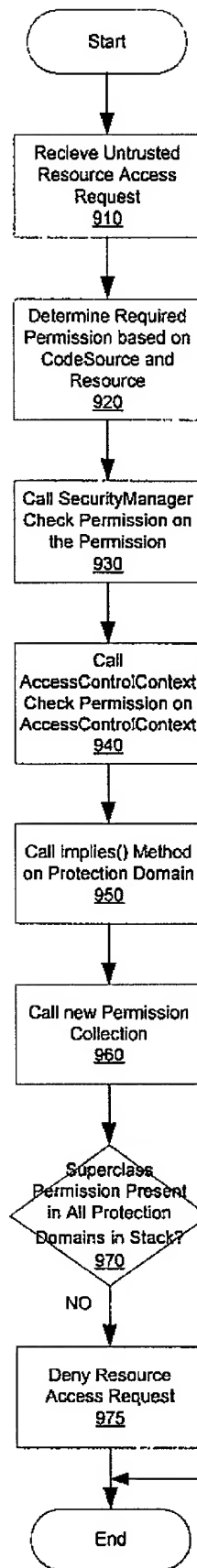
            String lineRead;
            while ((lineRead = br.readLine()) != null)
                System.out.println(lineRead);
        }

        catch(Exception e)
        {
            e.printStackTrace();
        }
    }
}

```

## Figure 8

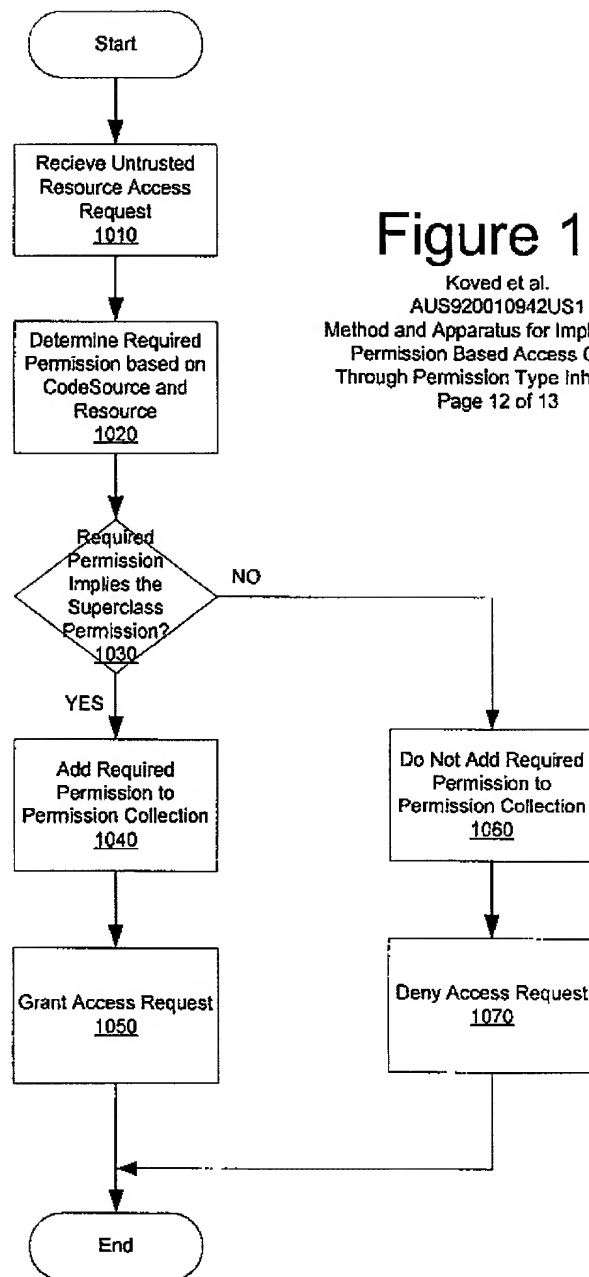
Koved et al.  
 AUS920010942US1  
 Method and Apparatus for Implementing  
 Permission Based Access Control  
 Through Permission Type Inheritance  
 Page 10 of 13



**Figure 9**

Koved et al.  
 AUS920010942US1  
 Method and Apparatus for Implementing  
 Permission Based Access Control  
 Through Permission Type Inheritance  
 Page 11 of 13

FIG. 9 is a flowchart illustrating a method for implementing permission based access control through permission type inheritance. The method starts at a "Start" block, followed by a process block "Receive Untrusted Resource Access Request 910". This is followed by "Determine Required Permission based on CodeSource and Resource 920", then "Call SecurityManager Check Permission on the Permission 930", and "Call AccessControlContext Check Permission on AccessControlContext 940". Next is "Call implies() Method on Protection Domain 950", followed by "Call new Permission Collection 960". A decision diamond "Superclass Permission Present in All Protection Domains in Stack? 970" follows. If the answer is "YES", the flow proceeds to "Add Permission to Permission Collection 980", then "Add Any Subclass Permissions to Permission Collection 985", then "Add Permission Collection to AccessControlContext 990", and finally "Grant Resource Access Request 995". If the answer is "NO", the flow proceeds to "Deny Resource Access Request 975". Both paths lead to an "End" block.



**Figure 10**

Koved et al.  
AUS920010942US1  
Method and Apparatus for Implementing  
Permission Based Access Control  
Through Permission Type Inheritance  
Page 12 of 13

```

package sun.security.provider;

import java.security.PermissionCollection;
import java.security.CodeSource;
import IBMPermission;
import WSPermission;

public class MarcoPolicy extends PolicyFile
{
    public PermissionCollection getPermissions(CodeSource codesource)
    {
        PermissionCollection pc = super.getPermissions(codesource);

        if (pc == null)
            return null;

        if (pc.implies(new IBMPermission("PermissionTest")))
            pc.add(new WSPermission("PermissionTest"));

        return pc;
    }
}

```

## Figure 11

Koved et al.  
 AUS920010942US1  
 Method and Apparatus for Implementing  
 Permission Based Access Control  
 Through Permission Type Inheritance  
 Page 13 of 13